原创 **骨骼动画合批在UE4下的实现**

比奥(王少标) 发布于 G121工作圈

发布时间:2021.05.11 10:26 593 15 0 更多

已推荐到：热门推荐

> 之前在Unity下实现过骨骼动画合批(AnimationInstancing)的
> 多，因为需要和底层的渲染管线紧密结合，会插入比较多的

1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

一、原理介绍

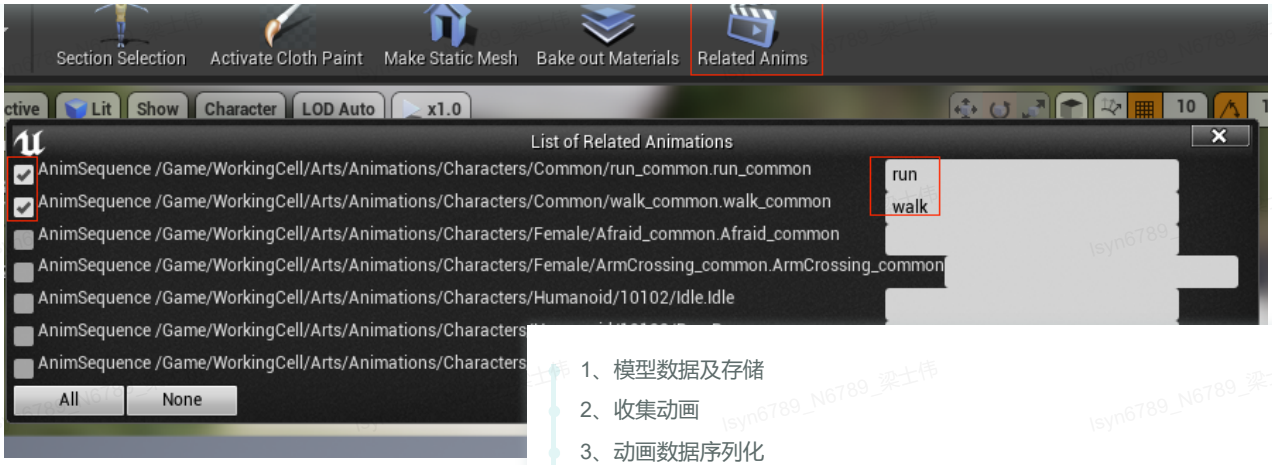
在传统的骨骼动画渲染中，每个带骨骼的模型都是单独渲染，这样如果想支持场景中同屏渲染成千上万个骨骼模型的话，会需要很多drawcall开销，导致帧率很低。为了解决这个问题，出现了AnimationInstancing（骨骼动画合批）技术，该技术主要解决骨骼动画模型的合批渲染问题。其基本原理是预先将骨骼动画烘焙到一张贴图上面（本文称为动画贴图），然后在渲染时的vertex shader阶段，根据当前模型的动画帧信息（属于哪个动画，在该动画的哪一帧），从动画贴图中采样得到需要的骨骼变换信息（gles2.0不支持在vertex shader阶段采样贴图），并结合模型的顶点属性（如position,normal等信息）及世界变化矩阵等，渲染出当前骨骼动画模型。 ~~~~

二、UE4下的实现细节

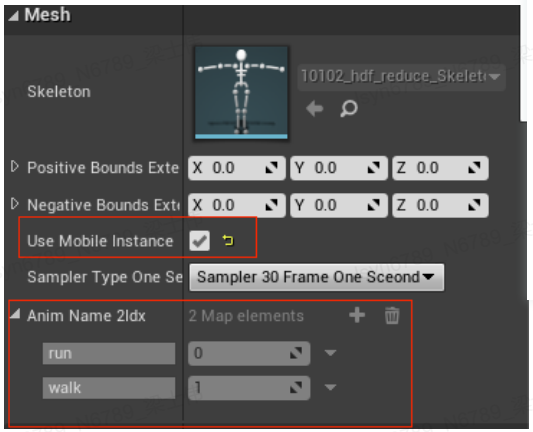
项目源码可通过https://km.netease.com/tech_share/project/875申请。

1、模型数据及存储

一般是会将SkeletalMesh转化为StaticMesh的形式，然后需要将骨骼索引及权重等信息转存到StaticMesh的顶点属性中，最终在StaticMesh上做instancing，原始的SkeletalMesh会被抛弃掉。G107战锤采用的就是上述这种形式。但是我这里没有采用上述这种方式，而是直接在SkeletalMesh上做Instancing。因为我们项目中，原始的SkeletalMesh可能会用引擎那套动画系统来控制动画（非合批模式）。直接将Instancing做在原始的SkeletalMesh，



接着勾选Use Mobile Instance即可自动为当前SkeletalMesh生成



上图AnimName2Idx是为了调试方便显示出当前烘焙的动画名。

1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

2、收集动画

需要将骨架和该SkeletalMesh一样的动画收集起来，代码如下：

```
FAssetRegistryModule& AssetRegistryModule = FModuleManager::LoadModuleChecked<FAssetRegistryModule>(TEXT("AssetRegistry"));
TArray<FAssetData> AnimAssetData;
AssetRegistryModule.Get().GetAssetsByClass(UAnimSequence::StaticClass()->GetFName(), AnimAssetData, true);
TArray<UAnimSequence*> AnimSequenceData;
for (auto& Temp : AnimAssetData)
```

```
        AnimSequenceData.Add(pAnimSequence);
        AnimDataMap.Add(FAnimtionDataMap(pAnimSequence->GetFName()));
    }
}
```

3、动画数据序列化

收集完动画数据后，通过FArchive支持将数据连同USkeletalMesh的其它数据一起序列化，以便存储和加载使用。

```
// 需要在USkeletalMesh加入如下AnimDataMap来存储动画数
UPROPERTY()
TArray<FAnimationDataMap> AnimDataMap;

//保存动画关键帧数据类
USTRUCT(BlueprintType)
struct ENGINE_API FAnimationDataMap
{
        GENERATED_USTRUCT_BODY()
        FAnimtionDataMap()
        {}
        FAnimtionDataMap(const FName& InAnimName)
        {
                AnimName = InAnimName;
        }
        friend FArchive& operator<<(FArchive& Ar, F
        UPROPERTY()
        FName    AnimName; //动画名称
        UPROPERTY()
        TArray<uint8> AnimRawData;//动画关键帧压缩得数据
        void Decompress();// const FReferenceSkeleton* RefSkeleton);//解压数据
        TMap<int32, TArray<FTransform>> BoneMapKey;//动画关键帧
};
```

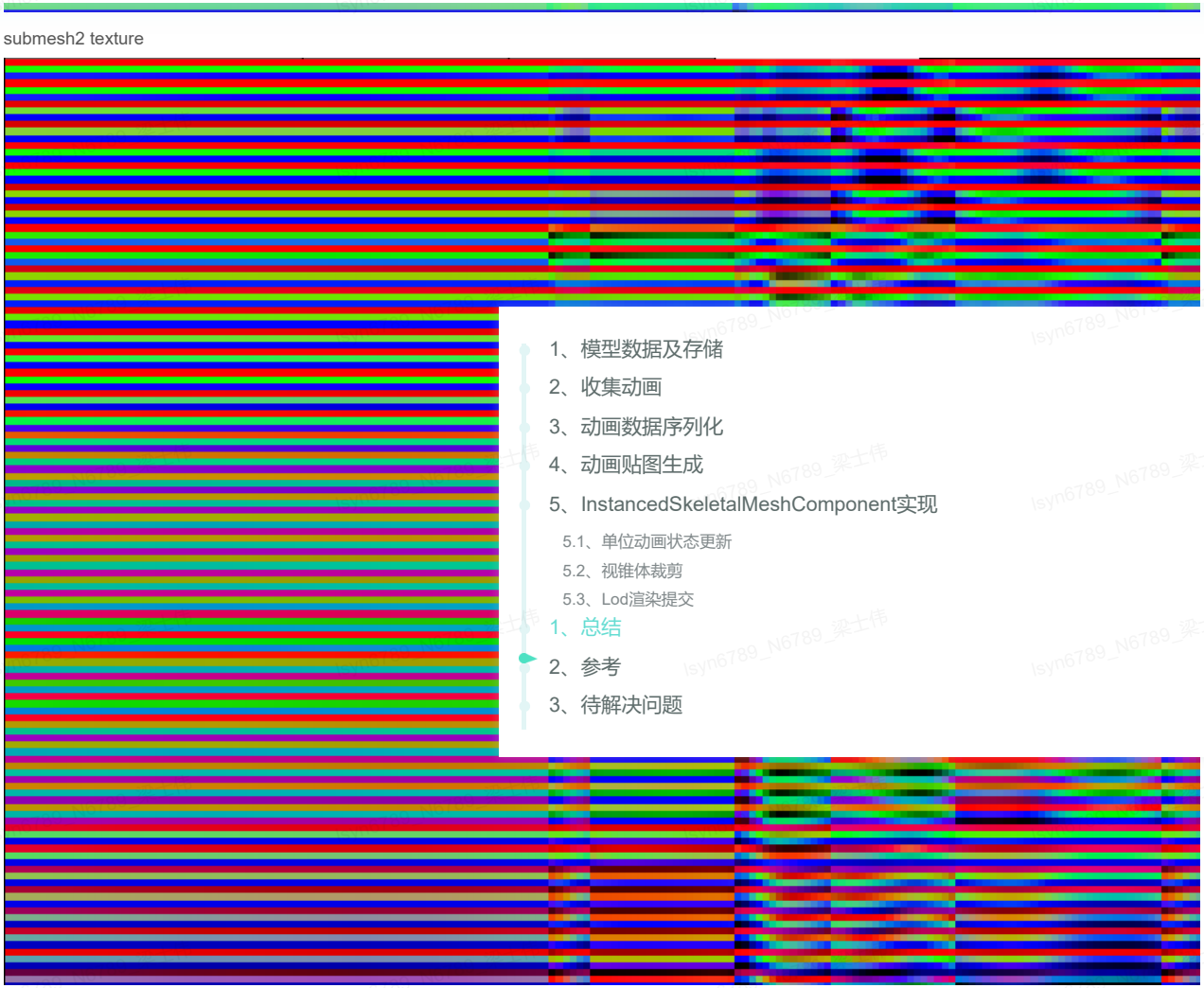

1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

4、动画贴图生成

首先会有个总的骨架动画数据，在InitResources时，各个lod的submesh根据自己用到的骨骼索引，去总的骨架动画里面获取并生成自己的子骨架动画贴图。为了减少贴图，需要限制lod的submesh数量为1-2个，并限制各级lod用的骨骼索引集需要一致。下面两个图显示了一个模型的子网格所用的骨骼集不一样时，会有不一样的动画贴图。

当然，我们还可以修改SkeletalMesh的顶点数据，让所有子网格的索引都用总的那个骨骼的。但考虑到我们的SkeletalMesh可能会用于普通的Animator功

submesh2 texture



1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

5 5、InstancedSkeletalMeshComponent实现

UE4里面自带了一个InstancedStaticMeshComponent组件，用于处理大量静态物体的手动合批。相比于每个Instance一个Component控制的方法，其好处就是将合批直接在生成MeshBatch之前手动做好，没必要等到MeshDrawCommand阶段再去合批，因为那样需要将MeshDrawCommand顺序进行比对来判断，当场景有大量相同物体时，这个比较判断还是比较耗时的。另外每个模型一个Component的方式在性能上也是很有问题的。所以这里我参照InstancedStaticMeshComponent实现了一个InstancedSkeletalMeshComponent，来统一处理同类模型的大量Instance更新渲染。

需要在TickComponent中每帧更新每个单位的动画状态，并调用MarkRenderDynamicDataDirty触发渲染更新。

```
void UInstancedSkeletalMeshComponent::TickComponent(......)
{
    ......
    for (int32 idx = 0; idx < NumUnits; ++idx)
        {
                FAnimationInstancingUnitData& UnitData = UnitDatas[idx];
                if (!UnitData.Used)
                        continue;

                FVector UnitPos(UnitData.WorldTrans
                //视锥体裁剪
                UnitData.IsVisible = ConVexVolume.I
                if (!UnitData.IsVisible)
                        continue; //不可见则跳过后续

                bool IsChangeAni = false;
                check(UnitData.AniIndex >= 0 && Uni

                int32 TotalFrame = UnitData.AniInde
                        SkeletalMesh->KeyFrameLeng
                        SkeletalMesh->KeyFrameLeng
>KeyFrameLengthArrray[UnitData.AniIndex - 1];
                // 动画切换
                if (UnitData.AniIndex != UnitData.P
                {
                        IsChangeAni = true;
                        UnitData.TransitionDuration = 0.5; // 切换时间
                        UnitData.TransitionProgress = 0.0; // 切换进度
                        UnitData.IsInTransition = true;
                        UnitData.PreAniFrame = UnitData.CurFrame;
                        UnitData.TransitionTimer = 0.0;
                        UnitData.CurFrame = 0.0;
                }
                // 动画过渡 更新
                if (UnitData.IsInTransition)
                {
                        UnitData.TransitionTimer += DeltaTime; // 已过渡时间
                        float Weight = UnitData.TransitionTimer / UnitData.TransitionDuration;
                        UnitData.TransitionProgress = FMath::Min(Weight, 1.0f); // 计算当前动画权重
                        if (UnitData.TransitionProgress >= 1.0) // 过渡完成判断
```

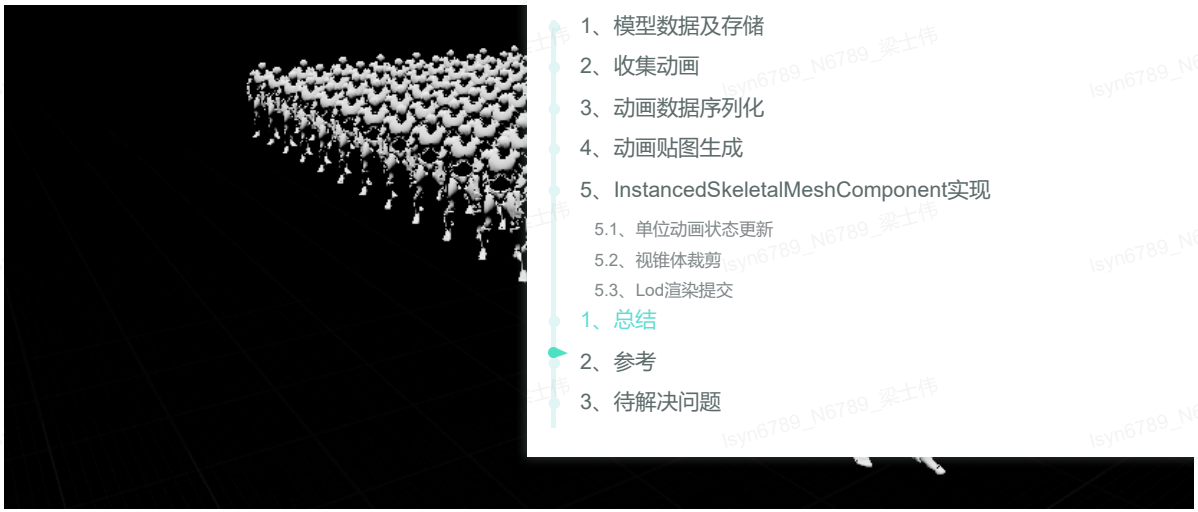

1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

```
                }
            }
            // 动画帧更新
            if (!IsChangeAni)
            {
                float PreFrame = UnitData.CurFrame;
                UnitData.CurFrame += DeltaTime * Fps;// TODO clip.fps;

                bool Looping = true;

                if (Looping)
                {
                    if (UnitData.CurFr
                        UnitData.C
                    else if (UnitData.
                        UnitData.C
                }
                UnitData.CurFrame = FMath:
            }

            // 计算Lod
            if (ComputeLod)
            {
                float Dist2Camera = (Camer
                if (Dist2Camera > LodLevel
                    UnitData.LodLevel
                else if (Dist2Camera > LodLevels[0])
                    UnitData.LodLevel = 1;
                else
                    UnitData.LodLevel = 0;
                UnitData.LodLevel = FMath::Clamp(UnitData.LodLevel, 0, 2); // TODO
            }
            else
            {
                UnitData.LodLevel = 0;
            }
        }
        // update all animationData
        MarkRenderDynamicDataDirty();
    }
```


1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

```
FSceneViewProjectionData ProjectionData;
LocalPlayer->GetProjectionData(LocalPlayer->ViewportClient->Viewport, eSSP_FULL, ProjectionData);
FMatrix ViewProjectionMatrix = ProjectionData.ComputeViewProjectionMatrix();
FConvexVolume ConVexVolume;
// 获取主相机视锥体的六个面
GetViewFrustumBounds(ConVexVolume, ViewProjectionMatrix, true);
// 判断当前Unit是否与视锥体相交，为了方便计算，这里用的包围盒都是球形。
UnitData.IsVisible = ConVexVolume.IntersectSphere(UnitPos + BoundingOffset, BoundingRadius);
```

裁剪效果如下：



1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

5.3、Lod渲染提交

由于同个模型的全部Instance都是在一个InstancedSkeletalMeshComponent下控制提交渲染的，原始的渲染管线对于每个Component只会生成当前Lod所有SubMesh的MeshBatch。而我们的Component会同时存在多级Lod模型，因此需要对MeshBatch的生成进行改造，让其能同时对Instance数量不为零的Lod都生成MeshBatch。核心代码如下：

```
void FSkeletalMeshSceneProxy::GetMeshElementsConditionallySelectable(.....) const
{
        if (IsMobileInstance())
        {
                int32 MaxLodLevel = FMath::Min<int32>(SkeletalMeshRenderData->LODRenderData.Num(),
MobileInstanceLodLevelOffsetArray.Num());
                for (int32 LODIndex = 0; LODIndex < MaxLodLevel; LODIndex++)
```

```
                        LodInstances = NumMobileInstances - MobileInstanceLodLevelOffsetArray[LODIndex];
                }
                else
                {
                        LodInstances = MobileInstanceLodLevelOffsetArray[LODIndex+1] -
MobileInstanceLodLevelOffsetArray[LODIndex];
                }
                // 当前Lod的Instance数量为0就跳过，可以减少一个空提交
                if (LodInstances <= 0)
                {
                        continue;
                }
                const FSkeletalMeshLODRend

                if (LODSections.Num() > 0 &
                {
                        ......
                        GetDynamicElements
SectionIndex, bSectionSelected, SectionElementInfo,
                }
        }
        return;
}
```


1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

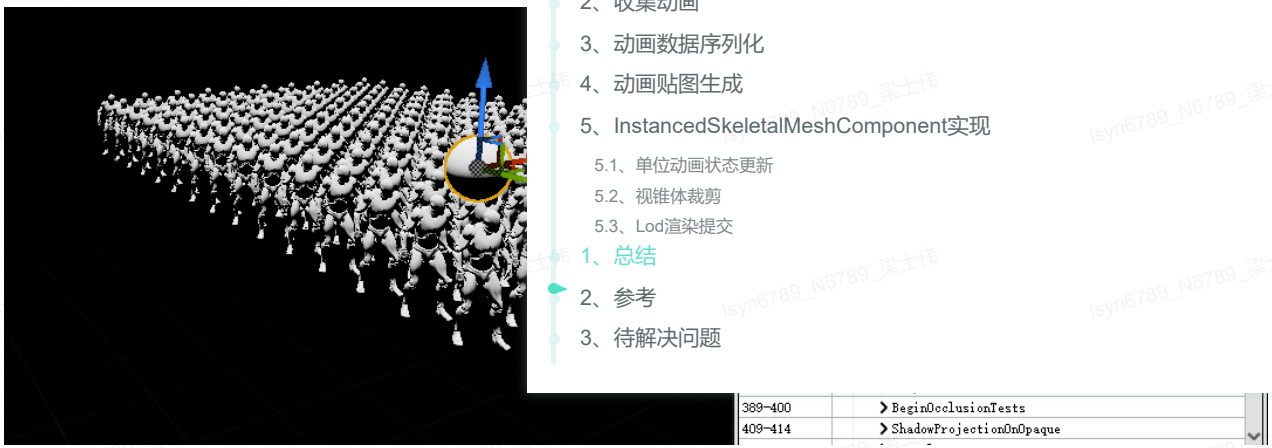
另外需要处理的一个问题就是，需要把每个Lod的Instancing Uni
Offset直接传某个Lod的Instancing UniformBuffer。
InstancedSkeletalMeshComponent会首先根据Instance的数量申请一大块显存MobileInstanceVertexBuffer，用于存放Instancing信息（Transform,Color等）。我的做法是把Instance根据其lod层级在这块显存上顺序存放，并用一个offset记录每级lod的起始显存索引，在各级lod最终提交渲染时，传入各自offset即可。代码如下：

```
void FMeshDrawCommand::SubmitDraw(......)
{
    ......
    for (int32 VertexBindingIndex = 0; VertexBindingIndex < MeshDrawCommand.VertexStreams.Num(); VertexBindingIndex++)
        {
                const FVertexInputStream& Stream = MeshDrawCommand.VertexStreams[VertexBindingIndex];

                // animation instancing
                if (MeshDrawCommand.InstanceIdStreamIndex != -1 && Stream.StreamIndex ==
MeshDrawCommand.InstanceIdStreamIndex && MeshDrawCommand.MobileInstanceVertexBuffer != nullptr)
                {
```

```
                else if (MeshDrawCommand.PrimitiveIdStreamIndex != -1 && Stream.StreamIndex ==
MeshDrawCommand.PrimitiveIdStreamIndex)
                {
                        RHICmdList.SetStreamSource(Stream.StreamIndex, ScenePrimitiveIdsBuffer, PrimitiveIdOffset);
                        StateCache.VertexStreams[Stream.StreamIndex] = Stream;
                }
                ......
        }
    ......
}
```


1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

lod渲染效果如下，图中共三级lod，他们各自都能用一个批次渲



三、总结

1、总结

上面只是介绍了接入过程中解决的主要问题，实际上还有很多细节没讲到，总共需要改到几十个源文件。UE4这个渲染管线给人的感觉就是框架比较清晰（SkeletalMesh->SceneProxy->MeshBatch->MeshDrawCommand->SubmitDraw），但是细节比较复杂，实现过程中需要在各个文件中跳来跳去地改，很容易绕晕。

2、参考

（1）目前动画更新都是在一个TickComponent里面统一做的，因为每个Instance的更新都是独立的，后面可以考虑加入多线程的更新。

1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题

全部评论 0

请输入评论内容

(可添加1个视频+5张图片) 匿名 评论

最热 最新

暂无评论

加载完毕,没有更多了

GM19总结

正当防卫系列中的地形系统和角色运动技术

UE5 Nanite 浅析 (一) : 核心思路

1、模型数据及存储
2、收集动画
3、动画数据序列化
4、动画贴图生成
5、InstancedSkeletalMeshComponent实现
5.1、单位动画状态更新
5.2、视锥体裁剪
5.3、Lod渲染提交
1、总结
2、参考
3、待解决问题